ARMY RESEARCH LABORATORY

Applying Distributed Computing to an EM Application in a UNIX Environment

by Brian B. Luu

ARL-TR-958

May 1996

19960617 095

Approved for public release; distribution unlimited.

[illegible]

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** May 1996	**3. REPORT TYPE AND DATES COVERED** Final, Sept.1993 to Oct.1994

4. TITLE AND SUBTITLE Applying Distributed Computing to an EM Application in a UNIX Environment	**5. FUNDING NUMBERS** PE: 62120
6. AUTHOR(S) Brian B. Luu	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory Attn: AMSRL-WT-MD 2800 Powder Mill Road Adelphi, MD 20783-1197	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TR-958
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory 2800 Powder Mill Road Adelphi, MD 20783-1197	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES AMS code: 612120.1400011 ARL PR: 4FE7E3	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

A distributed computing technique (DCT) in a network of 15 UNIX workstations is described and benchmark times given for a particular electromagnetic (EM) analysis application. The results demonstrate that the technique substantially improved the turnaround time for execution compared to a single processor system. Extension to an arbitrary network of processors and other applications is discussed in general. The technique is easily implemented and should be considered when a network of processors with a file server capability is available.

14. SUBJECT TERMS distributed computing, computer performance, UNIX, file server	**15. NUMBER OF PAGES** 30
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

Contents

Appendices

Figure

Table

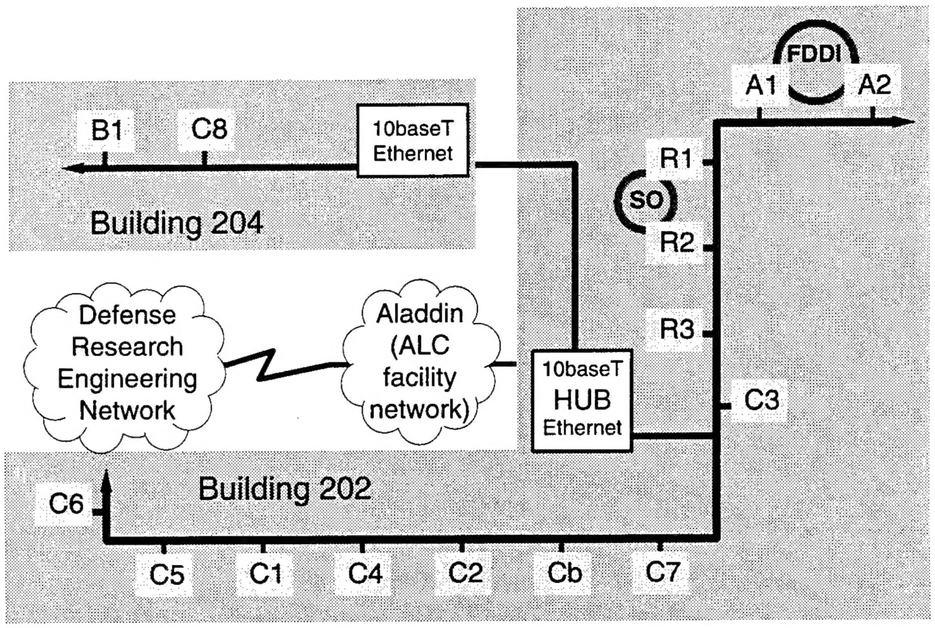
1. Introduction

The improvement of microprocessor speed and network connectivity has made data processing on microprocessor workstations more efficient and economical. With the maturity of operating systems and network software, it becomes feasible to distribute the required processing to individual workstations or personal computers that are networked together to achieve concurrent processing and reduce turnaround time. This report describes a distributed computing technique (DCT) employed in a UNIX system environment.

2. Environment

First, let's examine the system architecture in question, which accommodates distributed computing. Fifteen RISC (reduced instruction set chip) workstations make up the hardware. They are configured and tuned for engineering, scientific, and graphic applications. As depicted in figure 1, all workstations are networked together by an Ethernet™ local area network (LAN) operating with a data rate of 10 Mbits/s and a gateway to the Adelphi Local Area Digital Data Interchange Network (ALADDIN) at the Army Research Laboratory (ARL) and to the Defense Research Engineering Network (DREN). The workstations include three SGI Power IRIS 4D/440 systems (with four processors each), nine SGI Personal IRIS 4D/35 systems, two IBM Power Stations RS/6000-560, and one IBM Power Station RS/6000-530. This hardware makes up the main part of the system, called the Electromagnetic Effects Modeling System (EeMS), which was designed

Figure 1. Electromagnetic Effects Modeling System (EeMS).



FDDI = fiber distributed data interface
SO = serial optical channel converter network

to provide fast engineering workstations at the desks of certain scientists and engineers (S&Es) at ARL. All workstations run variations of the UNIX™ operating system (SGI IRIX™ and IBM AIX™) and use the TCP/IP protocol for networking. (Both IRIX and AIX are derivatives of AT&T UNIX and Berkeley UNIX.)

The total disk space in the EeMS is 50 gigabytes. Using the Network File System (NFS™) software and with a special arrangement of file systems, users can access their files from any system in the EeMS with the same file path. For example, say a user (user1) has a data file on the "emsa1" system with the file path /home/emsa1/user1/datafile: even when logged into another system (say, "emsc5"), that user can access that data file on emsa1 with the same file path: /home/emsa1/user1/datafile.

Automount, a feature of NFS, is used to reduce network traffic. The automount feature will mount remote file systems when they are needed. When the file systems are not used for a specific time (specified by the system administrator), automount will unmount the remote file systems. With this arrangement, users' application programs can access all users' files in the EeMS using the same file path from any system in the EeMS.

Network Information Service (NIS™) software is employed to provide every user a unique user identification (uid) on every workstation in the EeMS. This special arrangement of unique uids for users of the EeMS has alleviated problems associated with authentication and authorization of reading, writing, and executing users' files and programs but still maintained the necessary security for the operation of the EeMS.

3. Technique

By taking advantage of services provided by NFS and NIS software and the special design and arrangement of file systems, the computer environment of the EeMS enables and facilitates distributed processing among all EeMS processors. A distributed computing technique was implemented to concurrently process information on all workstations of the EeMS. The technique requires the following components:

- data pool file,
- index file, and
- getind function (for providing an index for the next available data set).

3.1 Data Pool File

The data pool file is a collection of the input data needed by DCT processes to compute a result. A process is a program submitted to a processor for execution. The input data should be grouped into sets (or records) and must be quantifiable. Each data set is the smallest amount of input data needed to produce a result. The data pool file must be accessible to all DCT processes, either through a local file system or a remote file system.

3.2 Index File

The index file contains the index number of the last used data set in the data pool file, as well as flags to signal all DCT processes or certain DCT processes at certain hosts to terminate. The index file must also be accessible to all DCT processes, either as a local file or through a network-remote file. While the index file is being accessed, the calling process must lock the index file by using a software lock mechanism, such as NFS lock, a feature provided in the UNIX NFS. When the calling process locks the index file, it has the exclusive right to write and modify the index file. When another calling process tries to lock an index file that is already locked, the calling process will be put on hold until a predefined time is up, and then it will terminate.

An example of the contents of the index file is

```
124 0
emsa1.arl.mil 3
emsc2.arl.mil 1
```

The first number of the first line is the index of the last used data set in the data pool file. The second number on the same line is the all-process-terminating flag, which signals all processes to terminate. The value of 1 for the all-process-terminating flag means "terminate," and the value of 0 means "proceed." Any lines after the first line are the host-process-terminating flags, which have the format of a hostname and number of processes. The process that is executed by the host system matching the hostname of the host-process-terminating flags will decrement by 1 the number of processes in this line and terminate. The number of processes can be greater than 1, because some host systems can execute more than one process on the DCT. For example, the emsa1 system has four processors, so it can launch four processes to execute four data sets at the same time. The index file can be manually modified so that the terminating flags are set. During the manual modification process, the index file should be locked.

3.3 getind Function

The purpose of the getind function is to provide the index to the next available data set in the data pool file and also to interpret the terminating flags set in the index file. The getind function will first try to lock the index file. Each time it fails to lock the index file, it will wait for a preset period and then try to lock again. If the number of tries exceeds the preset number, the getind function will quit and signal the process to terminate. After successfully locking the index file, the getind function checks the last used index for end-of-record and terminating flags. If all the terminating conditions are negative or not applicable to the calling process, then the getind function will increase the index value to the next available index value, update the index file, and return the next available index value to the calling process. Upon returning to the calling process, the getind function unlocks the index file.

An example of the input parameters of the getind function, listed in appendix A, is as follows:

- the maximum number of input data sets,
- the index file name,
- the hostname of the calling process.

Written in C language, this getind function used the "include" files unistd.h and fcntl.h. The getind function will try to lock the index file 25 times. Each time it fails, it will sleep for 5 s or less and retry again, for a total of 25 attempts (this can be varied depending on the user's requirements). If a hostname of the calling process appears on any of the host-process-terminating flags, then the getind function will decrement by 1 the number of processes and return a flag value of –2, to inform the calling process to terminate. After this decrement, if the number of processes is 0, then the getind function will remove this line. The getind function will return the value of the index of the next available data set, a value of 0 (if the index number reaches the maximum index number), or a value of –1 (if the all-process-terminating flag value is 1).

4. Algorithm

For implementation, this distributed computing technique requires the following arrangement:

- The index file and data pool file must be accessible to the DCT processes. The application program should be able to determine the number of data sets contained in the data pool file.
- There must be a software lock mechanism, such as NFS lock, to inform or prevent other processes from modifying the index file while a process is using it. The file lock mechanism should work across the network and also in a heterogeneous system environment, consisting of systems from different vendors.
- The application program should be structured in such a way that each run of a data set yields a result. The execution of one data set should be independent of the execution of any other data set in the data pool.

In order to obtain an input data set for execution, the application program will invoke the getind function to obtain the index of the next available data set in the data pool file. To use the DCT, the user will start or submit the application program on all available systems.

5. Performance, Problems, and Discussion

5.1 Performance

In an effort to speed up turnaround time, the technique described here was incorporated into an application program (listed in app A) used to compute the electric and magnetic field of a point in space radiated by an electromagnetic pulse (EMP) simulator.[1] In this case, a data set is the coordinate of the point in space. The DCT used all processors in the EeMS: 12 processors in three SGI 4D/440 systems, 9 processors in nine SGI 4D/35 systems, 2 processors in two IBM RS/6000-560 systems, and 1 processor in an IBM RS/6000-530 system. All parts of this program were written in C programming language.

Table 1 summarizes and allows comparison of the performance for this specific EM application using the DCT. Column 2 refers to the LINPACK benchmark for the normal mode of system operation. The Mflops benchmark is based on the LINPACK code of 200×200 array elements. The LINPACK code was linpack.c, retrieved from the netlib.att.com machine on the Internet. The LINPACK benchmark in C programming language was used, since the EM application was written in the C language (for more details, see app B). The execution time required to calculate the EM fields for one data set is shown in column 3. The average execution time for one run is based on the average execution time of three typical observation points. All the execution times referred to here are based on the normal operation of systems in the EeMS and assume that few or no other users' processes were using the systems beside these distributed computing processes. See appendix C for more details. Column 4 shows the application benchmark times normalized to the performance of the SGI 4D/35 system, the slowest system in the EeMS. The estimated execution time required to calculate the EM fields at 1000 observation points is shown in the last column. For this particular application, use of the DCT provides a reduction in execution time of almost a factor of 10 compared to an IBM RS/560.

Table 1. Summary of benchmarks and comparisons.

System	LINPACK benchmark (Mflops)	Average execution time for 1 data set (hh:mm:ss)	Comparison to SGI 4D/35 system	Estimated execution time for 1000 data sets (days)
SGI 4D/35	3.88	10:39:25	1.00000	445
SGI 4D/440	4.33	9:44:28	1.09402	406
IBM RS/530	11.69	7:39:45	1.39079	320
IBM RS/560	22.80	3:47:52	2.80610	159
15 systems with DCT	N/A	N/A	29.13120	16

[1] *Brian B. Luu and Calvin D. Le, AESOP Field Prediction, Army Research Laboratory, ARL-TR-835 (November 1995)*

5.2 Problems

For the heterogeneous system environment, the data format incompatibility of the index and data pool file can prevent systems from different vendors from correctly reading and processing the data sets. For example, DECstation 5000 can read the index file (since the index file is in ASCII), but it cannot correctly read the data in the data pool file, because the data pool file is in the binary floating point format for the SGI and IBM systems in the EeMS. A common data format (e.g., ASCII) for all systems should be used for the data pool file, the index file, and also the result data file. Otherwise, special input/output functions must be written for an application program to handle incompatible data formats.

During the course of execution, the systems that provide file server service for the index and data pool file must be operating at all times to provide the indexes and data sets for processes to execute. A power failure will disrupt the distributed computing process—especially a power failure to systems that provide the NFS services for the index and data pool file. The terminating flags in the index file can be used to properly terminate the processes if a power shutdown is expected or planned. If a power failure improperly terminates the distributed computing processes, the information in the index file will be used to restart the DCT process at the last unprocessed index data sets. Indexing of input data along with the DCT has saved the execution time that would be required to rerun data sets already completed. Using uninterruptible power supplies (UPSs) can mitigate unexpected power failures for systems. Note that the UPS for systems that provide the network services (such as the bridge, router, and NFS file servers for the index and data pool file) should outlast the UPSs for other systems.

5.3 Discussion

If all the results are not urgently needed, a lower priority can be set for the DCT processes so that the impact on other users' jobs is minimized. With a low execution priority status, the DCT processes will be put in a waiting queue or use a low proportion of CPU cycles compared to other users' processes with normal priority. But in the evening, when there are few or no other users' jobs running, the DCT processes will use all the available CPU cycles.

With the termination controls in the index file, the DCT offers flexibility for participating systems. Not all 15 systems of the EeMS have to be active at all times for the DCT. Some systems can be released from the DCT and reactivated at a later time. More systems participating in the DCT will result in shorter completion time for all the input data sets.

To alter the course of the DCT (e.g., terminating some or all distributed computing processes) without incurring improper termination, a lead time is needed. An adequate lead time requires an amount of time that is equal to or greater than the amount of execution time for one data set on the slowest system. But this required lead time can be longer, depending on the CPU load of the slowest system at that time.

Many factors contribute to an uncertainty in the prediction of the completion time of the DCT process. The uncontrollable factors are CPU load, network speed, network traffic, and unexpected events (e.g., power failure, system failure). But the amount of execution time for one data set on the slowest system is the determining factor in this uncertainty.

For maximum benefit from the DCT, the minimum number of data sets involved should be greater than the sum of the performance times normalized to the performance time of the slowest processor (as illustrated in column 4 of table 1) of the participating processors. Obviously, the minimum number of data sets should be greater than the number of processors. For example, for the maximum benefit of using DCT on all systems in the EeMS, the minimum number of data sets in this application should be 29. But for the DCT performed on one processor of an SGI 4D/35, one processor of an SGI 4D/440, one processor of an IBM RS/530, and one processor of an IBM RS/560, the minimum number of data sets should be 6.

6. Conclusion

As implemented and tested, this distributed computing technique has demonstrated its utility for applications in which a large task can be divided into many small tasks, and each small task executed independently on any available system on the network. Usually, these large applications, as in electromagnetics or acoustics, require supercomputer capability, which has very limited flexibility. The distributed computing technique is suitable for a system environment consisting of microprocessor workstations networked together. This type of system environment is now emerging in the industry: for example, networking of personal computer (PC) Pentium workstations that are equipped with high-level operating systems and network software. This environment is inexpensive and flexible, requires less system administration, and eliminates the chance of single focal operation failure, which could happen on a centralized system environment like a mainframe or multiprocessor supercomputer. The fundamental requirements to implement the distributed computing technique are a network of systems and file server capability.

Appendix A. Application Program Listing

```
/* program clob.c
   date: 24 Jan 1995
   Author: Brian B. Luu
   Description: This C programming code will read in XYZ coordinates of points
in space and compute the estimated electric and magnetic field radiated by AESOP
(Army Electromagnetic Pulse Simulator Operation) at these locations.
                The coefficents of the current distribution on each dipole segment
are precomputed and saved in a file which is hardcoded in the program with the
file name "/home/emsc3/bluu/aesop/CUCOEF.DATA".
                All the input and output formats of the program are in SGI-IRIX or
IBM-AIX binary-floating-point format except the "data pool" file which is in ASCII.

*/

#include <stdio.h>
#include <math.h>

#define MAXT     2000
#define MAXEH   16008
#define MAXS    14978
#define MAXSH   10040
#define MAXCO  299580
#define MAXS2   29956

double cuc, cup;
double co[MAXCO];
double tcrs, sn0, sn1, rn0, rn1;

main(int argc, char *argv[]) /* Main program */
{
    /* Initialized data */

    char spc;
    char dscuco[]="/home/emsc3/bluu/aesop/CUCOEF.DATA";
    char findex[100], hn[64];
    char dseht[160]="mkdir ";

    struct { double hdd[24]; int hdi[8]; } hdr;

    int i, ib;
    int it, its;
    int is, isb, isl, isgx, isgz;
    int isg[4];
    int *itm=hdr.hdi;
    int index, nds, nrun=0;

    double c=3.e8;
    double dt=1.e-9;
    double dlmin=1.e-2;
    double al=20.;
    double oxt=100.4, ozt=al;
```

```
     double sax=47.5, saz=13.5;
     double cucc=2.73448e-13;
     double cul=4.07e-9;
     double cur=.005364;
     double dl, zoo4pi,  pi;
     double cosa, sina, sa;
     double eh[MAXEH];
     double rs[4], rl[4], ros[4], rot[4];
     double tr, tehs, retard, etc, htc, cu, cud, cvo, tcs;
     double *ro=&hdr.hdd[1];
     double *ehm=&hdr.hdd[16];
     double air, rlp2, rlp3, xor, xorp2;

     register double *dp;
     FILE *fp, *fpd;

     size_t sizeof_double=sizeof(double);
     size_t sizeof_eh=sizeof(eh);
     size_t sizeof_hdr=sizeof(hdr);

     extern void exit(int);

     extern int current(int , double, double *, double *, double *);
     extern int getind(int, char *, char *);

     cup = sqrt(cul * cucc);
     cuc = cur / (sqrt(cul / cucc) * 2.);

/*   INITIALIZE PARAMETERS */

     pi = atan(1.) * 4.;
     zoo4pi = c * 1e-7;
     rs[1] = 0.;
     sa = sqrt(sax * sax + saz * saz);
     cosa = sax / sa;
     sina = saz / sa;

/*   OBTAIN X&Z SWITCH DATAPOOL FILENAME AND INDEX FILENAME   */

    if (argc < 4)
    {
       printf("Not enough data: no switch number or seperate character or \
datapool filename.\n");
       exit(3);
    }
    else
    {
       switch (argv[1][0])
       {
          case '1':
          {
             isg[0] =  1;
             isg[1] =  1;
          }
```

```
            break;
            case '2':
            {
               isg[0] = -1;
               isg[1] = -1;
            }
            break;
            case '3':
            {
               isg[0] =  1;
               isg[1] = -1;
            }
            break;
            default:
            {
               printf("%c: invalid input for x switch; \
must be 1, 2, or 3\n", argv[1][0]);
               exit(1);
            }
         }
         switch (argv[1][1])
         {
            case '1':
            {
               isg[2] =  1;
               isg[3] =  1;
            }
            break;
            case '2':
            {
               isg[2] = -1;
               isg[3] = -1;
            }
            break;
            case '3':
            {
               isg[2] =  1;
               isg[3] = -1;
            }
            break;
            default:
            {
               printf("%c: invalid input for z switch; \
must be 1, 2, or 3\n", argv[1][1]);
               exit(1);
            }
         }
         air = (argv[1][0] - '0')*10 + (argv[1][1] - '0');

         spc = argv[2][0];
         strcat(strcpy(findex, argv[3]), ".ind");
         if ((fp=fopen(findex, "r")) == NULL)
         {
            if ((fp=fopen(findex, "r")) == NULL)
            {
                fp=fopen(findex, "w");
                fprintf(fp, "%10d %5d\n", 0, 0);
            }
         }
         fclose(fp);
      }

/*  READ IN CURRENT COEFFICIENTS OF ALL SEGMENTS */

      fp = fopen(dscuco,"r");
      for (i = 1, dp = co+20; i <= MAXS; ++i, dp += 20) {
          fscanf(fp, "%*d %le %le %le %le %le %le %le %le %le %le \
                          %le %le %le %le %le %le %le %le %le %le",
           dp, dp+1, dp+2, dp+3, dp+4, dp+5, dp+6, dp+7, dp+8, dp+9,
           dp+10, dp+11, dp+12, dp+13, dp+14, dp+15, dp+16, dp+17, dp+18, dp+19);
      }
      fclose(fp);

/*  GET HOSTNAME                                          */

     gethostname(hn, 64);

/*  READ PARTITION DATA SET NAME OF THE E AND H FIELD */

     if ((fpd=fopen(argv[3],"rb")) == NULL)
     {
            printf("Cannot open datapool file: %s;  Program terminated.\n", argv[3]);
            exit(2);
     }

     fread((void *) (dseht+6), sizeof(char), 80, fpd);
     fseek(fpd, 0L, SEEK_END);
     nds = (ftell(fpd) - 80)/(13*sizeof_double);
     ib = (int) strlen(dseht);

/*  READ IN SEGMENT LENGTH AND COORDINATE OF OBSERVATION POINT */

     while ((index=getind(nds, findex, hn)) > 0)
     {
        fseek(fpd, 80 + (index-1)*13*sizeof_double, SEEK_SET);
        fread((void *) hdr.hdd, sizeof_double, 13, fpd);

        dl = hdr.hdd[0];
        isl = (dl + dlmin / 2.) / dlmin;

        printf("%10d:%g(%+20.13e,%+20.13e,%+20.13e)\n", index, dl, ro[0],
                  ro[1], ro[2]);
        fflush(stdout);

/*  DETERMINE THE STARTING TIME */

        isb = (isl + 1) >> 1;
        isgx = 1;
      if (ro[0] < 0.) isgx = -1;
      rs[0] = isgx * ((isb << 1) - 1) * (dlmin / 2.);
      rlp2 = ro[0] - rs[0];
      rlp3 = ro[2] - al;
      tehs = sqrt(rlp2 * rlp2 + ro[1] * ro[1] + rlp3 * rlp3) / c + isb * cup;

  /*  INITIALIZE EH ARRAY */

      for (dp = eh, i = 0; i < MAXEH; ++i) *dp++ = 0.;

  /*  CALCULATE E AND H FIELD */

      for (is = isb; is <= MAXSH; is += isl) {
         tcs = is * cup;
         tcrs = ((MAXS - is << 1) + 1) * cup;
         sn0 = is - 1;
         sn1 = is;
         rn0 = MAXS2 - (is - 1);
         rn1 = MAXS2 - is;
         for (isgz = isg[2]; isgz >= isg[3]; isgz += -2) {
            for (isgx = isg[0]; isgx >= isg[1]; isgx += -2) {
               rs[0] = isgx * ((is << 1) - 1) * (dlmin / 2.);
               rs[2] = isgz * al;
               rl[0] = ro[0] - rs[0];
               rl[1] = ro[1] - rs[1];
               rl[2] = ro[2] - rs[2];
               rl[3] = sqrt(rl[0] * rl[0] + rl[1] * rl[1] + rl[2] * rl[2]);
               retard = rl[3] / c + tcs - tehs;
               its = retard / dt + 1.;
               for (it = its; it <= MAXT; ++it) {
                  tr = it * dt - retard;
                  current(is, tr, &cu, &cud, &cvo);

                  rlp2 = rl[3] * rl[3];
                  rlp3 = rl[3] * rlp2;
                  xor = rl[0] / rl[3];
                  xorp2 = xor * xor;
                  dp = &eh[it*8];

                  etc = isgz * zoo4pi * dl * (cud / c / rl[3] + cu / rlp2 *
                              3. + c * cvo / rlp3 * 1.5 );
                 *dp++ += isgz * zoo4pi * dl * (cud / c / rl[3] *
                             (xorp2 - 1.) + (cu / rlp2 +
                          cvo / 2. * c  / rlp3) * (xorp2 * 3. - 1.) );
                 *dp++ += rl[1] / rl[3] * xor * etc;
                 *dp++ += rl[2] / rl[3] * xor * etc;

                 htc = isgz * dl * (cud / c / rl[3] + cu / rlp2)/(pi * 4.);
                ++dp;
                 *dp++ -= rl[2] / rl[3] * htc;
                 *dp   += rl[1] / rl[3] * htc;
               }
            }
         }
```

```
        }

/*  CALCULATE THE FIELD CONTRIBUTE BY THE SLANT PARTS */

        rs[2] = 0.;
        rot[1] = ro[1];
        for (; is <= MAXS; is += isl)
        {
           tcs = is * cup;
           tcrs = ((MAXS - is << 1) + 1) * cup;
           sn0 = is - 1;
           sn1 = is;
           rn0 = MAXS2 - (is - 1);
           rn1 = MAXS2 - is;
           for (isgz = isg[2]; isgz >= isg[3]; isgz += -2)
           {
               for (isgx = isg[0]; isgx >= isg[1]; isgx += -2)
              {
                  ros[0] = ro[0] - isgx * oxt;
                  ros[2] = ro[2] - isgz * ozt;
                  rot[0] = cosa * ros[0] - isgx * isgz * sina * ros[2];
                  rot[2] = isgx * isgz * sina * ros[0] + cosa * ros[2];
                  rs[0] = isgx * ((is - MAXSH << 1) - 1) * (dlmin / 2.);
                  rl[0] = rot[0] - rs[0];
                  rl[1] = rot[1] - rs[1];
                  rl[2] = rot[2] - rs[2];
                  rl[3] = sqrt(rl[0] * rl[0] + rl[1] * rl[1] + rl[2] * rl[2]);

                  retard = rl[3] / c + tcs - tehs;
                  its = retard / dt + 1.;
                  for (it = its; it <= MAXT; ++it)
                 {
                     tr = it * dt - retard;
                     current(is, tr, &cu, &cud, &cvo);

                     rlp2 = rl[3] * rl[3];
                     rlp3 = rl[3] * rlp2;
                     xor = rl[0] / rl[3];
                     xorp2 = xor * xor;
                     dp = &eh[it*8];

                     etc = isgz * zoo4pi * dl * (cud / c / rl[3] + cu / rlp2 *
                                 3. + c * cvo / rlp3 * 1.5 );
                     ehm[0] = isgz * zoo4pi * dl * (cud / c / rl[3] *
                                (xorp2 - 1.) + (cu / rlp2 +
                                cvo / 2. * c  / rlp3) * (xorp2 * 3. - 1.) );
                     ehm[2] = rl[2] / rl[3] * xor * etc;
                     *dp++ += cosa * ehm[0] + isgx * isgz * sina * ehm[2];
                     *dp++ += rl[1] / rl[3] * xor * etc;
                     *dp++ -= isgx * isgz * sina * ehm[0] + cosa * ehm[2];

                     htc = isgz * dl * (cud / c / rl[3] + cu / rlp2)/(pi * 4.);
                     ehm[5] = rl[1] / rl[3] * htc;
                     *dp++ += isgx * isgz * sina * ehm[5];
```

```
                    *dp++ -= rl[2] / rl[3] * htc;
                    *dp   += cosa * ehm[5];
                }
            }
        }
    }

/*  DETERMINE THE MAX VALUES OF E & H FIELDS AND THEIR TIMES */

    for (i = 0; i < 8; ++i)
    {
        ehm[i] = 0.;
        itm[i] = 0;
    }

    for (dp = eh, it = 0; it <= MAXT; ++it)
    {
        dp[6] = sqrt(dp[0] * dp[0] + dp[1] * dp[1] + dp[2] * dp[2]);
        dp[7] = sqrt(dp[3] * dp[3] + dp[4] * dp[4] + dp[5] * dp[5]);
        for (i = 0; i < 6; ++dp, ++i)
        {
            if (fabs(*dp) > fabs(ehm[i]))
            {
                ehm[i] = *dp;
                itm[i] = it;
            }
        }
        for (i = 6; i < 8; ++dp, ++i)
        {
            if (*dp > ehm[i])
            {
                ehm[i] = *dp;
                itm[i] = it;
            }
        }
    }

/*  SET A DATA SET NAME FOR E AND H FIELD */

    sprintf(dseht+ib,"%cz%+20.13e%cy%+20.13e%cx%+20.13e", spc, ro[2],
            spc, ro[1], spc, ro[0]);

    memmove((void *) hdr.hdd, (void *) &hdr.hdd[1], 6*sizeof_double);
    memmove((void *) &hdr.hdd[8], (void *) &hdr.hdd[7], 6*sizeof_double);
    hdr.hdd[6] = dl;
    hdr.hdd[7] = dt;
    hdr.hdd[14] = tehs;
    hdr.hdd[15] = air;

/*  WRITE E AND H FIELD TO FILE */

    if ((fp=fopen(dseht+6,"wb")) == NULL)
    {
          dseht[ib+44] = '\0';

/*               check if y directory is created or not                */

          if ((fp=fopen(dseht+6,"r")) == NULL)
          {
             dseht[ib+22] = '\0';

/*               check if z directory is created or not                */

             if ((fp=fopen(dseht+6,"r")) == NULL)
             {
                dseht[ib] = '\0';

/*               check if base directory is created or not           */

                if ((fp=fopen(dseht+6,"r")) == NULL)
                {
                   if (system(dseht))        /* create base directory          */
                   {
                      printf("Cannot create directory: %s; \
 Program terminated.\n", dseht+6);
                      exit(1);
                   }
                }
                else
                   fclose(fp);         /* base directory is already creater */
                dseht[ib] = spc;
                if (system(dseht))        /* create z directory          */
                {
                   printf("Cannot create directory: %s; \
Program terminated.\n", dseht+6);
                   exit(1);
                }
             }
             else
                fclose(fp);             /* z directory is already created */

             dseht[ib+22] = spc;

             if (system(dseht))          /* create y directory            */
             {
                printf("Cannot create directory: %s;  Program terminated.\n",
                           dseht+6);
                exit(1);
             }
          }
          else
          {
               dseht[ib+44] = spc;
               printf("Cannot create file: %s;  Program terminated.\n",
                            dseht+6);
               exit(1);
          }
```

```
            dseht[ib+44] = spc;
            fp = fopen(dseht+6,"wb");
         }

         fwrite((void *) &hdr, sizeof_hdr, 1, fp);
         fwrite((void *) eh, sizeof_eh, 1, fp);
         fclose(fp);
         ++nrun;
         printf("%10d> %s\n", nrun, dseht+6);
      }

      fclose(fpd);
      switch (index)
      {
         case  0:
            printf("Program is successfully completed.\n");
         break;
         case -1:
            printf("All processes were instructed to terminate.\n");
         break;
         case -2:
            printf("Program was instructed to terminate.\n");
         break;
         case -3:
            printf("Cannot obtain index file: %s;  Program terminated.\n", findex);
         break;
         default:
            printf("Program abnormally terminated with index = %d\n", index);
      }
   }


   /* *   SUBROUTINE CALCULATE THE DIPOLE'S */
   /* *                         CURRENT, */
   /* *                         CURRENT DERIVATIVE, */
   /* *                         CONVOLUTION OF SIGN FUNCTION AND CURRENT */


   int current(int ns, double ts, double *y, double *yd, double *yco)
   {
        int lr;
        double ys, ysd, ysco, t;
        extern int curdis(int, int, double, double, double *, double *, double *);


        lr = 0;
        curdis(ns, lr, sn1, ts, &ys, &ysd, &ysco);
        t = ts + cup;
        curdis(ns, lr, sn0, t, y, yd, yco);
        *y -= ys;
        *yd -= ysd;
        *yco -= ysco;
```

```
/*  REFLECTION CURRENT IS NOT EMERGING */

     if ((t=ts-tcrs)<=0.) return 0;

/*  REFLECTION CURRENT */

     lr = 12;
     curdis(ns, lr, rn0, t, &ys, &ysd, &ysco);
     *y += ys;
     *yd += ysd;
     *yco += ysco;
     t += cup;
     curdis(ns, lr, rn1, t, &ys, &ysd, &ysco);
     *y -= ys;
     *yd -= ysd;
     *yco -= ysco;
     return 0;
} /* current */



/* *  SUBROUTINE CALCULATES CURRENT PARAMETERS  ** */


int curdis(int ns, int ir, double sn, double t,
                             double *ys, double *ysd, double *ysco)
{
     register double *dp;
     double aa, ab, ac, ad, pd, af;
     double ea, eb, ec, ed, edp,ef;
     double ar, er;

/*      AA  = CO( 0+IR,NS) */
/*      AB  = CO( 1+IR,NS) */
/*      AC  = CO( 2+IR,NS) */
/*      AD  = CO( 3+IR,NS) */
/*      PD  = CO( 4+IR,NS) */
/*      AF  = CO( 5+IR,NS) */
/*      EA  = CO( 6,NS) */
/*      EB  = CO( 7,NS) */
/*      EC  = CO( 8,NS) */
/*      ED  = CO( 9,NS) */
/*      EDP = CO(10,NS) */
/*      EF  = CO(11,NS) */
/*      RA  = CO(12,NS) */
/*      RB  = CO(13,NS) */
/*      RC  = CO(14,NS) */
/*      RD  = CO(15,NS) */
/*      PDR = CO(16,NS) */
/*      RF  = CO(17,NS) */
/*      AR  = CO(18,NS) */
/*      ER  = CO(19,NS) */
/*   CURRENT DISTRIBUTION */

     dp   = &co[ns*20];
     aa   = *(dp + 0 + ir);
     ab   = *(dp + 1 + ir);
     ac   = *(dp + 2 + ir);
     ad   = *(dp + 3 + ir);
     pd   = *(dp + 4 + ir);
     af   = *(dp + 5 + ir);
     ea   = *(dp + 6);
     eb   = *(dp + 7);
     ec   = *(dp + 8);
     ed   = *(dp + 9);
     edp  = *(dp +10);
     ef   = *(dp +11);
     ar   = *(dp +18);
     er   = *(dp +19);

     *ys   = exp(- sn * cuc) * ( aa * exp(ea * t) +
                                 ab * exp(eb * t) +
                                 ac * exp(ec * t) +
                                 af * exp(ef * t) +
                            2. * ad * exp(ed * t) * cos(edp * t + pd) );


/*   DERIVATIVE OF CURRENT DISTRIBUTION */

     *ysd = exp(- sn * cuc) * ( aa * exp(ea * t) * ea +
                                ab * exp(eb * t) * eb +
                                ac * exp(ec * t) * ec +
                                af * exp(ef * t) * ef +
                                2. * ad * exp(ed * t) *
                  (ed * cos(edp * t + pd) - edp * sin(edp * t + pd)) );

/*   CONVOLUTION OF SGN(T) AND I(T) */

    *ysco   = exp(- sn * cuc) * ( aa * (2. * exp(ea * t) -1.) / ea +
                                  ab * (2. * exp(eb * t) -1.) / eb +
                                  ac * (2. * exp(ec * t) -1.) / ec +
                                  af * (2. * exp(ef * t) -1.) / ef +
                             2. * ad * (2. * exp(ed * t) *
                    (ed * cos(edp * t + pd) + edp * sin(edp * t + pd) ) -
                     ed * cos(pd)           - edp * sin(pd)              ) /
                    (ed * ed + edp * edp) );


     if (ir == 0) return 0;
     *ys    += exp(- sn * cuc) * ar * exp(er * t);
     *ysd   += exp(- sn * cuc) * ar * exp(er * t) * er;
     *ysco += exp(- sn * cuc) * ar * (2. * exp(er * t) -1.) / er;
     return 0;
} /* curdis */
```

```
/*     FUNCTION  getind
                 TO OBTAIN INDEX FOR DATA POOL
*/

#include <unistd.h>
#include <fcntl.h>
#ifdef _H_FCNTL
#include <sys/lockf.h>
#define O_RDWR          2
#endif

int getind(int nds, char *fn, char *hn)
{
    struct { char name[64]; int ext;} hst[25];
    int fd, i,j, index, sflag, try;
    FILE *fp;

    try = 0;

    fp = fopen(fn, "r+");
    fd = fp->_file;
    while (lockf(fd, F_TLOCK, 0L) < 0)
    {
        if(++try > 26)
        {
            fclose(fp);
            return -3;
        }
        sleep((try>5)?5:try);
    }
    fscanf(fp, "%d %d", &index, &sflag);
    if (index >= nds)
        index = 0;
    else if (sflag)
        sflag = -1;
    else
    {
        i = 0;
        while (fscanf(fp, "%s %d", hst[i].name, &hst[i].ext) != EOF)
        {
            if (strcmp(hst[i].name,hn) == 0)
            {
                sflag = -2;
                if (–hst[i].ext == 0) –i;
            }
            ++i;
        }

        if (sflag == 0)
            ++index;

        freopen(fn, "w+", fp);
        fprintf(fp, "%10d %5d\n", index, 0);
        for (j=0; j < i; ++j)
```

```
            fprintf(fp, "%-64s %3d\n", hst[j].name, hst[j].ext);
        }
        fclose(fp);

        return sflag ? sflag : index;
    }
```

Appendix B. LINPACK Benchmarking of Workstations

The performance of processors in the Electromagnetic Effects Modeling System (EeMS) was benchmarked with the LINPACK benchmark. LINPACK is an industry benchmark that measures the floating-point performance of computer systems. The LINPACK benchmark in the C programming language was used, since this electromagnetic application was written entirely in the C language. The LINPACK code, linpack.c, was obtained from the netlib.att.com machine on the Internet. The benchmark used 15-digit double precision (8-byte representation) and 200 by 200 array elements, which required 315 kbytes of system RAM (random access memory). The LINPACK benchmark was submitted to processors in batch mode during a period of light user activity. Tables B-1 to B-4 present the results of LINPACK benchmarking on four different types of workstations in the EeMS.

The tables also list the time percentage of the overhead and two main routines of the benchmark program, DGEFA and DGESL, in which the majority of floating-point operations are performed. The DGEFA function is used to factor a double precision matrix by the use of Gaussian elimination. The DGESL function is for solving the double precision system ($\mathbf{AX} = \mathbf{B}$ or $\mathbf{A}^T\mathbf{X} = \mathbf{B}$).

Table B-1. Average rolled and unrolled performance for an SGI 4D/35 system.

Reps	Time (s)	DGEFA (%)	DGESL (%)	Overhead (%)	Kflops
2	0.79	86.08	3.80	10.13	3868.545
4	1.58	87.34	3.80	8.86	3814.815
8	3.13	88.18	2.56	9.27	3868.545
16	6.31	87.16	3.01	9.83	3861.746
32	12.58	87.04	3.02	9.94	3878.788

Table B-2. Average rolled and unrolled performance for an SGI 4D/440 system.

Reps	Time (s)	DGEFA (%)	DGESL (%)	Overhead (%)	Kflops
2	0.71	85.92	2.82	11.27	4359.788
4	1.40	87.86	2.14	10.00	4359.788
8	2.83	87.28	2.83	9.89	4308.497
16	5.64	87.23	3.01	9.75	4316.961
32	11.29	86.71	3.28	10.01	4325.459

Table B-3. Average rolled and unrolled performance for an IBM RS/6000-530 system.

Reps	Time (s)	DGEFA (%)	DGESL (%)	Overhead (%)	Kflops
4	0.57	84.21	0.00	15.79	11444.444
8	1.15	75.65	7.83	16.52	11444.444
16	2.34	81.62	1.71	16.67	11268.376
32	4.61	82.21	1.74	16.05	11355.728
64	9.20	76.85	1.20	21.96	12241.411
128	18.50	78.49	2.81	18.70	11687.943

Table B-4. Average rolled and unrolled performance for an IBM RS/6000-560 system.

Reps	Time (s)	DGEFA (%)	DGESL (%)	Overhead (%)	Kflops
8	0.58	82.76	0.00	17.24	22888.889
16	1.15	82.61	0.87	16.52	22888.889
32	2.30	75.65	1.74	22.61	24689.139
64	4.61	80.48	3.47	16.05	22711.456
128	9.21	82.41	1.41	16.18	22770.294
256	18.40	79.08	4.73	16.20	22799.827

Appendix C. Application Benchmarking of Workstations

The electromagnetic (EM) application program named "clob" was benchmarked on the four types of workstations in the Electromagnetic Effects Measurement System (EeMS): SGI 4D/35, SGI 4D/440, IBM RS/6000-530, and IBM RS/6000-560. On each system, the benchmark program used the same input data sets, which were based on three typical observation points. The benchmark tests were executed during a period in which few or no other user's jobs were being run, such as at night or on the weekend. The total execution time on each system was computed in two ways: one based on the time stamp on files and the other based on the UNIX "timex" command. The average execution time for one data set is determined based on the total execution time of all three data sets. These two methods produced average execution times that were very close. In the analysis of the distributed computing technique (DCT), the average execution time based on the UNIX "timex" command was used, but the fractions of seconds were discarded.

Table C-1. EM application benchmarking recorded using file time stamp.

	Execution time of benchmark on			
Benchmark data (data point)	SGI 4D/35	SGI 4D/440	IBM RS/6000-530	IBM RS/6000-560
(0,50,20)	10:06:32	9:15:02	7:17:39	3:36:55
(1000,0,300)	11:10:54	10:13:34	8:01:16	3:58:33
(100,100,100)	10:40:50	9:45:19	7:40:22	3:48:08

Table C-2. Average execution time for one data set.

	Average execution time of benchmark on			
Computed based on	SGI 4D/35	SGI 4D/440	IBM RS/6000-530	IBM RS/6000-560
file time stamp	10:39:25.33	9:44:38.33	7:39:45.67	3:47:52.00
timex command	10:39:25.78	9:44:28.89	7:39:45.47	3:47:52.13

Distribution

Admnstr
Defns Techl Info Ctr
Attn DTIC-OCP
8725 John J Kingman Rd Ste 0944
FT Belvoir VA 22060-6218

Dir
Defns Intllgnc Agcy
Attn RTS-2A Techl Lib
Washington DC 20301

Defns Nuc Agcy
Attn RAEE Elect Effects Div
6801 Telegraph Rd
Alexandria VA 22310-3398

Cmdr
US Army ARDEC
Attn AMSTA-AR-AEC-IE N Svendsen
Attn AMSTA-AR-CCL-D W Williams
Bldg 65 N
Picatinny Arsenal NJ 07806-5000

US Army AVRDEC
Attn AMSAT-R-EFM P Haselbauer
4300 Goodfellow Blvd
ST Louis MO 63120-1798

US Army BRDEC
Attn SATB-FGE J Ferrick
Attn SATB-FGE T Childers
FT Belvoir VA 22060-5606

US Army Matl Cmnd
Attn AMCAM-CN
5001 Eisenhower Ave
Alexandria VA 22333-0001

Dir
US Army Mis Cmnd (USAMICOM)
Attn AMSMI-RD-CS-R Documents
Redstone Arsenal AL 35898-5400

Cmdr
US Army MRDEC
Attn AMSMI-RD-ST-CM J Vandier
Huntsville AL 35898-5240

US Army Natick RDEC
Attn SATNC-SUSD-SHD A Murphy
Kansas Stret
Natick MA 01760-5018

US Army Nuc & Chem Agcy
Attn MONA-NU R Pfeffer
Attn MONA-TS Lib
7150 Heller Loop Rd Ste 101
Springfield VA 22150

US Army TARDEC
Attn AMSTA-ZT G Baker
Warren MI 48397-5000

US Army TECOM
Attn STERT-TE-E J Knaur
Redstone Technical Test Center
Huntsville AL 35898-8052

US Army TECOM Techl Dir Ofc
Attn AMSTE-TC-D R Bell
Aberdeen Proving Ground MD 21005

Cmdr
US Army White Sands Missile Range
Attn STEWS-NE J Meason
White Sands Missile Range NM 88002-5180

Nav Rsrch Lab
Attn Code 4820 Techl Info Div
4555 Overlook Ave SW
Washington DC 20375-5000

Cmdr
Nav Surfc Weapons Ctr
Attn Code E231 Techl Lib
Dahlgren VA 22448-5020

Natl Inst of Stand & Techlgy
Attn V Ulbrecht Rsrch Info Ctr
Rm E01 Bldg 101
Gaithersburg MD 20899

Distribution

DoD Joint Spectrum Ctr
Attn CA J Word
120 Worthing Basin
Annapolis MD 21401

US Army Rsrch Lab
Attn AMSRL-SL-CM M Mar
Aberdeen Proving Ground MD 21005-5068

US Army Rsrch Lab
Attn AMSRL-OP-SD-TA Mail & Records Mgmt
Attn AMSRL-OP-SD-TL Tech Library (3 copies)
Attn AMSRL-OP-SD-TP Tech Pub (5 copies)
Attn AMSRL-WT-N Chf

US Army Rsrch Lab (cont'd)
Attn AMSRL-WT-NB Chf
Attn AMSRL-WT-ND B Luu (5 copies)
Attn AMSRL-WT-ND Chf
Attn AMSRL-WT-ND R J Chase
Attn AMSRL-WT-ND W O Coburn
Attn AMSRL-WT-NE Chf
Attn AMSRL-WT-NF Chf
Attn AMSRL-WT-NG Chf
Attn AMSRL-WT-NH Chf
Attn AMSRL-WT-NJ Chf
Attn AMSRL-WT-N Sr Rsrch Scntst
Attn AMSRL-SC-A S Choy
Adelphi MD 20783-1197